DØ note XXXX

A Fast Algorithm to Find Clusters for L2CPS

Qichun Xu

University of Michigan, Ann Arbor, MI 48105

Abstract -- **This note will descript a fast algorithm to find clusters in CPS data for DØ L2 trigger system and its detailed implementation on dzero trigger system. The algorithm is capable of finding all clusters with their width and address in about 65 clock ticks from a set of 1280 channels, which is all scinilator strip number of one end for a CPS layer. The algorithm is implemented in FPGAs (Field Programmable Gate Array) using VHDL language.**

I. BACKGROUND

The upgraded Dzero detector includes a silicon vertex detector, central track detector, PreShower detector, calorimeter and muon detector. There is also a superconductor solenoid to provide a central magnetic field of 2 Tesla. The upgraded trigger system for Run II will include 4 levels [1]. The first trigger (L0) provides an inelastic collision trigger. It is implemented in hardware. The second trigger (L1) is implemented in a firmware and requires either a minimum ET deposition in the calorimeter or primitive tracks of minimum pT in the muon chambers. The third trigger (L2) is a mixture of firmware and software. It provides further rejection to the event rate by considering the correlations of trigger information from different detectors. The fourth and last trigger (L3) is wholly software trigger system implemented on a farm of computers that performs a nearly complete reconstruction of the events. The final event rate stored on tape will be less than 10 Hz.

In RunII at Dzero experiment, the new PreShower detector plays an important role in the electron identification. It also helps to restore the electromagnetic energy resolution that is degraded by the presence of the superconductor solenoid [2].

The central PreShower consists of three cylindrical layers, as is shown in Fig 2. Each layer is constructed with a set of scintillating strips whose output is proportional to the energy deposit on them by a particle or shower of particles. The inner layer (Axial Layer) is built with scintillator strip parallel to the beam line. The outer two layers (U and V layer) are built with the same scintillator strips, which have a stereo angle of 23^0. The central PreShower detector is located between the solenoid and the calorimeter. The total number of strips is quite large. Each layer is made with 1280 scintillating strips and is divided into north and south halves. The total number of individual strips is 7680.

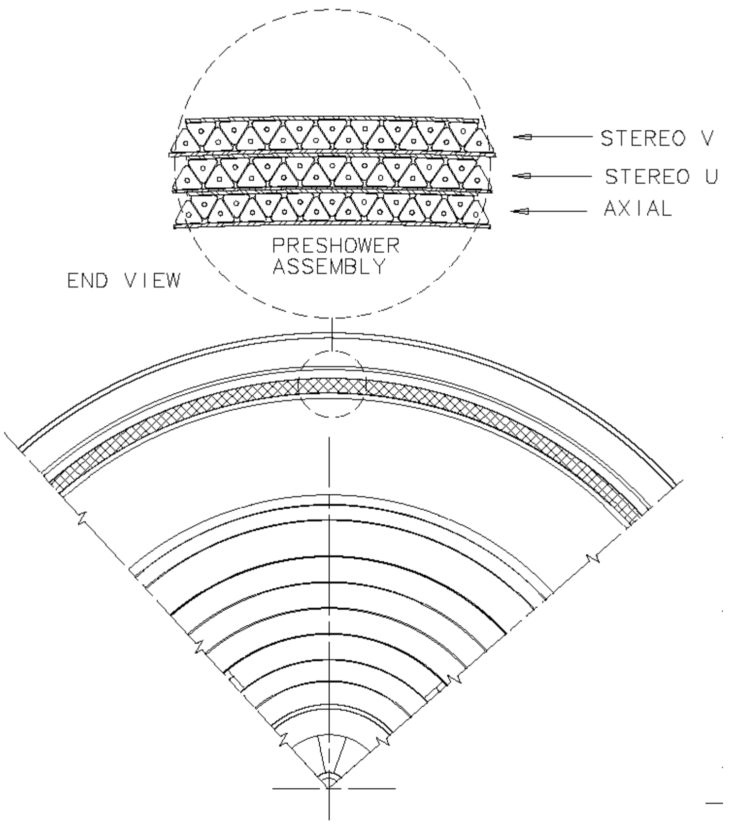


Fig. 2. The cross section view of central PreShower in Dzero detector.

In every collision, only a very small fraction of strips is activated. These active strips tend to be organized into blocks (called clusters) separated by large number of inactive strips. See Fig 3. These clusters have to be detected and reported to the trigger preprocessor in real time. Very fast processing is required so that the dead time is as small as possible. In Dezro, the data generated on the PreShower is first processed by an Analogy-Front-End board (AFE) and sent via high-speed serial links to a Digital-Front-End board (DFE). The DFE processes the data to find clusters and report them to the PreShower pre-processor.

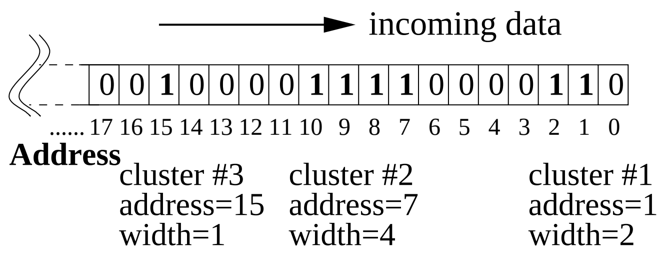


Fig. 3. The definition of a cluster is a block of data that is consecutive.

II. Algorithm

To detect clusters, one has to focus on the bit that is one. One naive algorithm is to start from the beginning of data and loop over all data stream, as shown in Fig 4. Each time data arrives, it is checked for the presence of a "hit". If a hit is detected, the data is either appended to an existing cluster or the start of a new cluster. A simple finite state machine can easily handle this algorithm.

Since this algorithm can only check one element at a time, in cases where large number of data needs to be processed in a short time, this method will not work well. In the case of the PreShower detector, to process data for a layer, it will take 1280 clock periods. At the clock frequency of 53MHz, the process will take more than 24 microseconds. This is far more than the allowed time for this operation, which is less than 2 microseconds. This is constrained by the time between signal L1_Accept arrives and all data are reported to pre-processors.

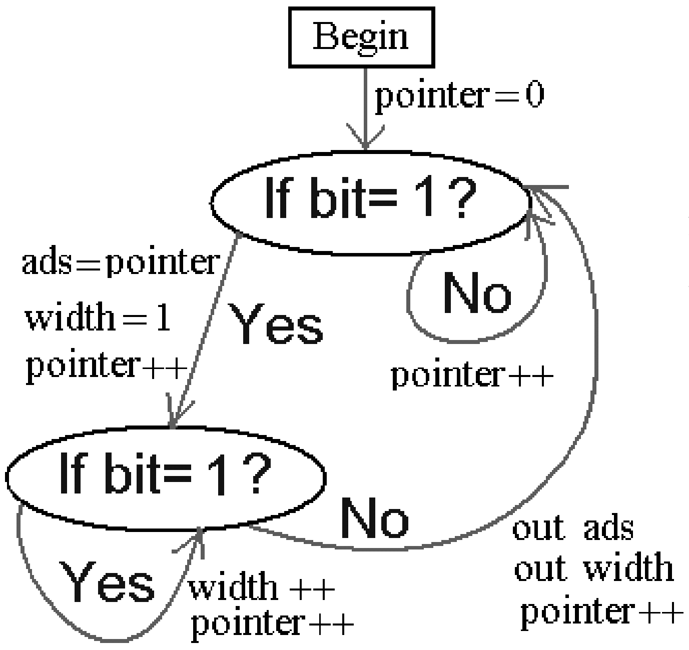


Fig. 4. Finite state machine (FSM) diagram for a simple algorithm to find clusters in data. At each clock, one bit of data is checked.

Although simple parallel processing could reduce the time required, it can not deal with subtle problems. For instance, as data is divided into several blocks, some clusters could be broken into two clusters by the edges of the blocks. To avoid this, an extra operation of cluster combination must be performed at later time. Also, parallel processing requires more FPGA resources to implement the algorithm. If we want to reduce the processing time by a factor of 10, the amount of resources needs to increase by a factor of 10. Obviously a compromise must be found.

Instead of one bit per clock cycle, we can manage to find an algorithm that processes several bits of data simultaneously. For the possible split of cluster that can occur, cluster combination is included.

In this algorithm, data arrives in parallel with n bit width. The central idea is to find the first cluster that exists in the data by pattern matching. The cluster is then “removed” from the incoming data. The same procedure is used for the rest of the data until no cluster remains. After all clusters are detected, they are compared to the previous data and cluster combination is performed if necessary. The cluster search and combination work in a pipeline. Only one extra clock cycle is needed for cluster combination.

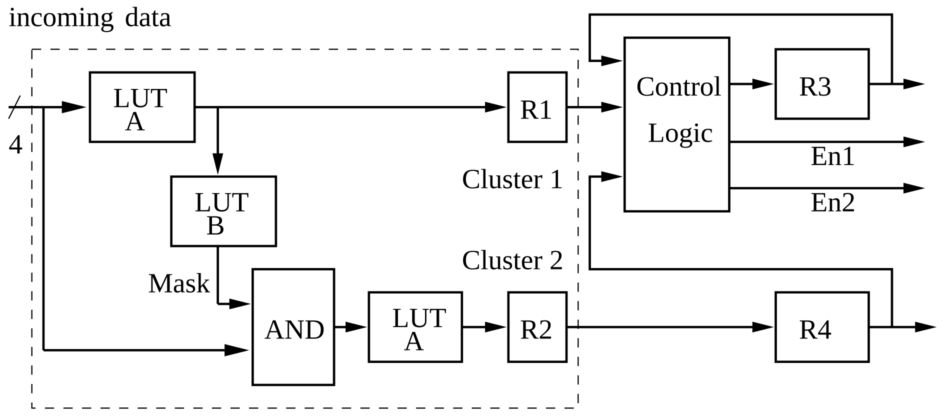


Fig. 5. Diagram shows a fast algorithm to find cluster in data for the case of data width equal to 4. There are tow kinds of look-up-tables (LUTs). LUT A finds the very first cluster in data by pattern matching. LUT B is to restore the cluster pattern. The Control Logic is used to combine clusters that could be separated in different incoming data block.

Principally, the data width n could be any integer. For convenience of implementation, we can take n to be a power of 2, such as 2, 4 or 8, etc. For n=4, there are at most 2 clusters in a group of incoming data and the diagram of the algorithms is shown in the Fig 5.

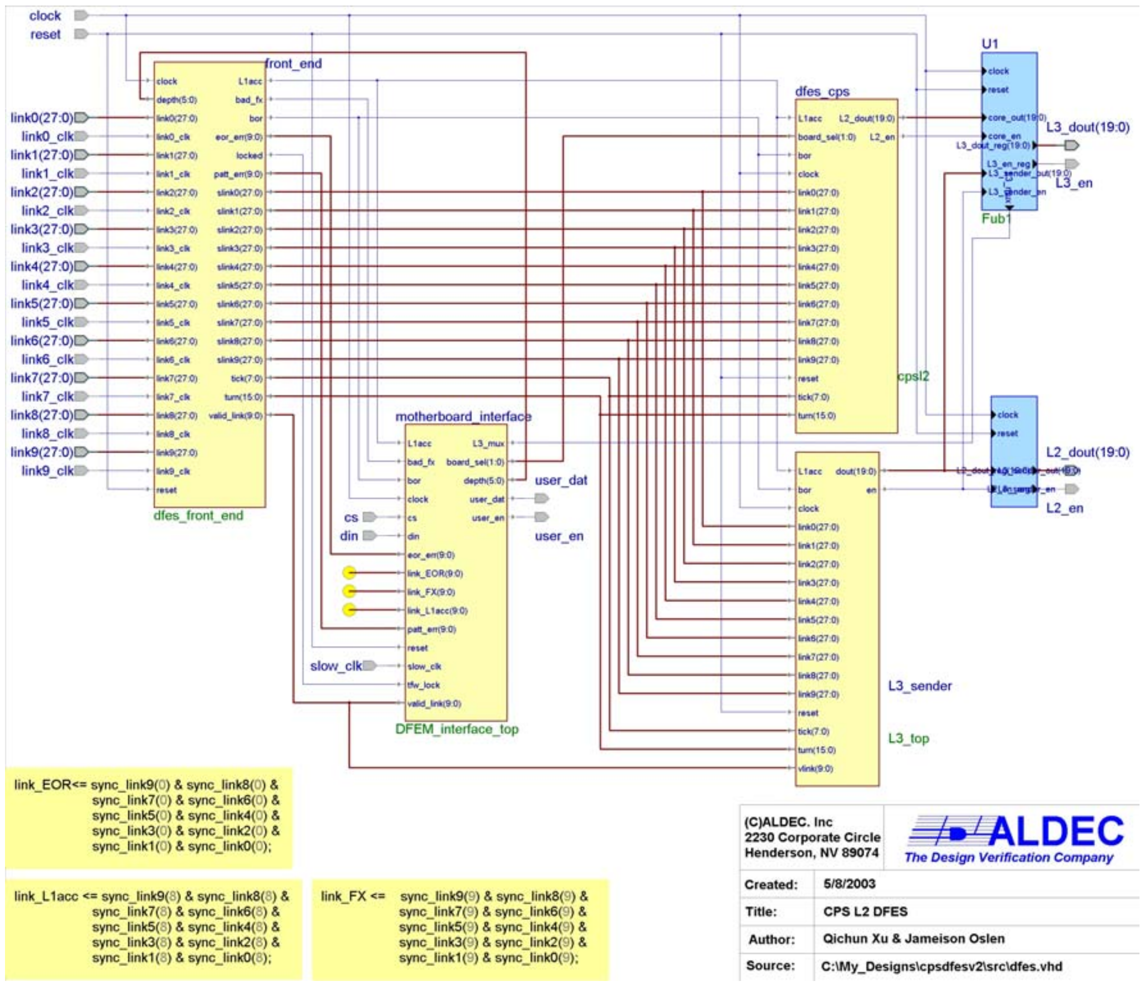
TABLE I TRUE VALUE TABLE OF LUT(LOOK UP TABLE)

Input	1st cluster				2nd cluster			
	A	W	B3	B0	A	W	B3	B0
0000	No cluster				No cluster			
0001	11	00	0	1	No cluster			
0010	10	00	0	0	No cluster			
0011	10	01	0	1	No cluster			
0100	10	00	0	0	No cluster			
0101	01	00	0	0	11	00	0	1
0110	01	01	0	0	No cluster			
0111	01	10	0	1	No cluster			
1000	00	00	1	0	No cluster			
1001	00	00	1	0	11	00	0	1
1010	00	00	1	0	10	00	0	0
1011	00	00	1	0	10	01	0	1
1100	00	01	1	0	No cluster			
1101	00	01	1	0	11	00	0	1
1110	00	10	1	0	No cluster			
1111	00	11	1	1	No cluster			

In the above example, 4 bits of data are processed simultaneously. Thus, the speed of this algorithm is increased by a factor of 4. This example can easily be extended to the cases where data width equals 8 or more.

In this case, the incoming data go through a look-up table (LUT A), which will decode it to find the relative address and width of the first cluster that may present. In this LUT, only the first cluster will be detected and 2nd cluster, if exists, is just ignored. The results are stored in a buffer and also go through another look-up-table (LUT B.) In LUT B, the pattern of first cluster is restored. An AND operation masks out the first cluster from the data. The remaining data goes through another LUT A. All these operation are finished within one clock cycle. The cluster information will be registered with the next clock. The control logic determines if cluster combination is needed and how to do it depending the present and previous data. The control logic also generates signals En1 and En2, which inform next stage if cluster data are coming.

For n=4, the procedure to find clusters could actually evolve into passing the data through a set of simple LUTs. The output of such LUTs is the information pertaining to the clusters that may exist within the data, such as widths and addresses. The function inside the dashed line in Fig.5 could be simplified into a decoding circuit, which is shown in Table I, where "A" and "W" stand for address and width respectively, "B3" and "B0" are the most significant bit (MSB) and least significant bit (LSB) of the corresponding cluster respectively. The purpose to have them is for the convenience of possible cluster combination in later stage.



```
link_EOR<= sync_link9(0) & sync_link8(0) &
           sync_link7(0) & sync_link6(0) &
           sync_link5(0) & sync_link4(0) &
           sync_link3(0) & sync_link2(0) &
           sync_link1(0) & sync_link0(0);

link_L1acc <= sync_link9(8) & sync_link8(8) &
              sync_link7(8) & sync_link6(8) &
              sync_link5(8) & sync_link4(8) &
              sync_link3(8) & sync_link2(8) &
              sync_link1(8) & sync_link0(8);

link_FX <=    sync_link9(9) & sync_link8(9) &
              sync_link7(9) & sync_link6(9) &
              sync_link5(9) & sync_link4(9) &
              sync_link3(9) & sync_link2(9) &
              sync_link1(9) & sync_link0(9);
```

(C)ALDEC. Inc
2230 Corporate Circle
Henderson, NV 89074

ALDEC
The Design Verification Company

Created:	5/8/2003
Title:	CPS L2 DFES
Author:	Qichun Xu & Jameison Oslen
Source:	C:\My_Designs\cpsdfesv2\src\dfes.vhd

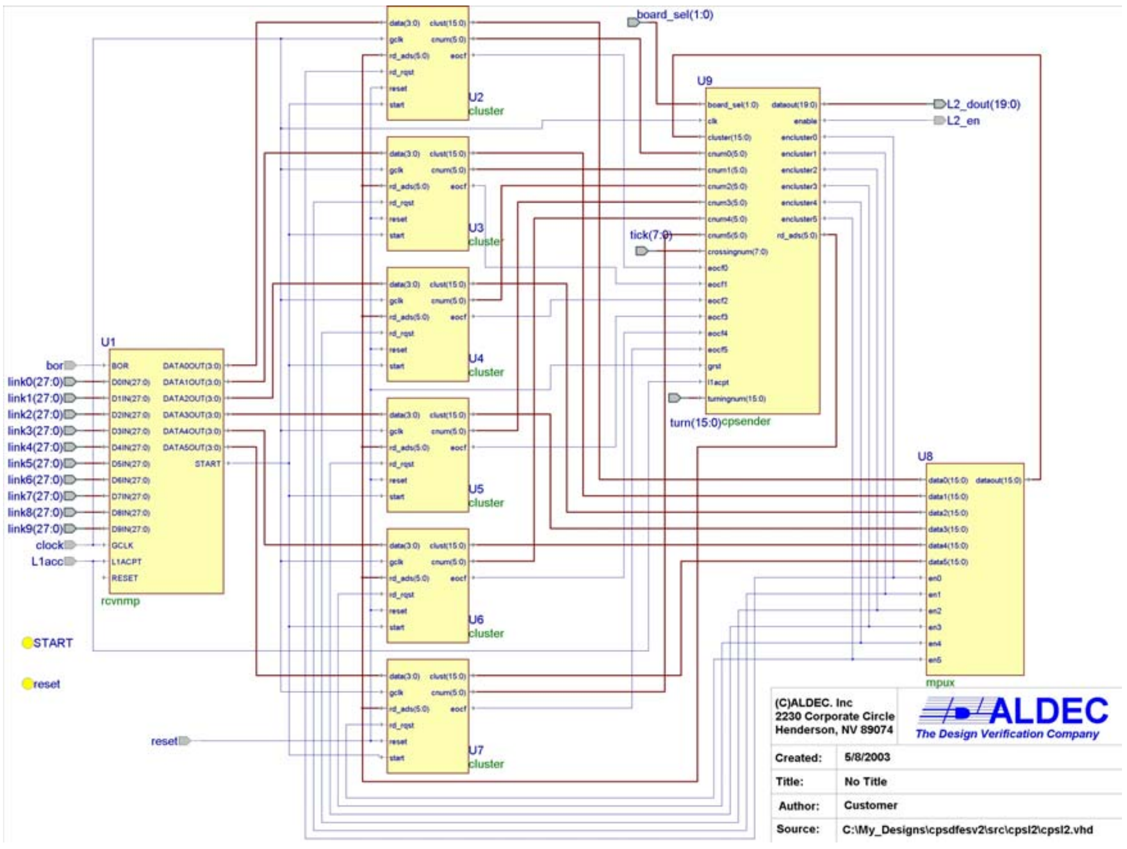
Fig. 6. Diagram of DFES implementation.

III. IMPLEMENTATION

Implementation with this algorithm are written in VHDL by using Xilinx Foundation© software and ALDEC Active-HDL. The FPGA used for L2CPS axial is VIRTEXE 1000BG560, also from Xilinx©.

The very top level is called CPSDFESVx (dfes.vhd) (the file name in parenthesis is the file name of corresponding VHDL code), which x corresponds to version number. It has four sub-levels, DFES_FRONT_END (dfes_front_end.vhd), L3_TOP (L3_top.vhd), DFEM_INTERFACE_TOP (DFEM_interface_top.vhd) and CPSL2 (cpsl2.vhd). The DFES_FRONT_END handles the synchronization of ten input links, the DFEM_INTERFACE_TOP works as interface between firmware and DFE motherboard and L3_TOP send raw data directly to L3. The CPSL2 is the core part of DFES to implements the physical mapping, cluster-finding algorithm and data transmitting to L2 through G-link. The details are shown in Figure 6. Except the CPSL2, the three other parts are written by Jameison Olsen. For details about this design, please see related document in his web site or contact him <jamieson@fnal.gov>. The following section will concentrate on the CPSL2.

There are mainly three parts in the design CPSL2, the receiver and mapping (rcvnmp.vhd), cluster finding (cluster_fnd.vhd) and data transmitting (mpux.vhd and l3_sender.vhd). The details are shown in Figure 7.



(C)ALDEC. Inc
2230 Corporate Circle
Henderson, NV 89074

ALDEC
The Design Verification Company

Created:	5/8/2003
Title:	No Title
Author:	Customer
Source:	C:\My_Designs\cpsdfesv2\src\cpsl2\cpsl2.vhd

Fig. 7. Diagram of CPSL2 implementation

After L1_ACCEPT is detected from input data stream in DFES_FRONT_END, ten-link data are sent to receiver and mapping, where data are re-arranged according to the data transferring protocol of AFE. Because of U/V mixing from each AFE, all data are divided into U layer data and V layer data. Each corresponds to half of U/V layer, which are 640 strips. This 640-bit data further separate into three parts, each with 256-bit of data. The third part only has 128-bit and the rest are filled with zeros. These 256-bit data are sent to the next module with 4-bit width within 64 clock periods. The next step is to find cluster in the data and put into memory. It also has signal of CNUM and EOCF, which are cluster number and end of cluster finding respectively. Together with cluster data, they go to the multiplex and data sender. The MPUX mix cluster data from six implementation of cluster finding. The l2_sender sends all cluster data out to L2 preprocessor according to protocol. The total cluster number is limited to 48. The clusters that are more than 48 will be truncated.

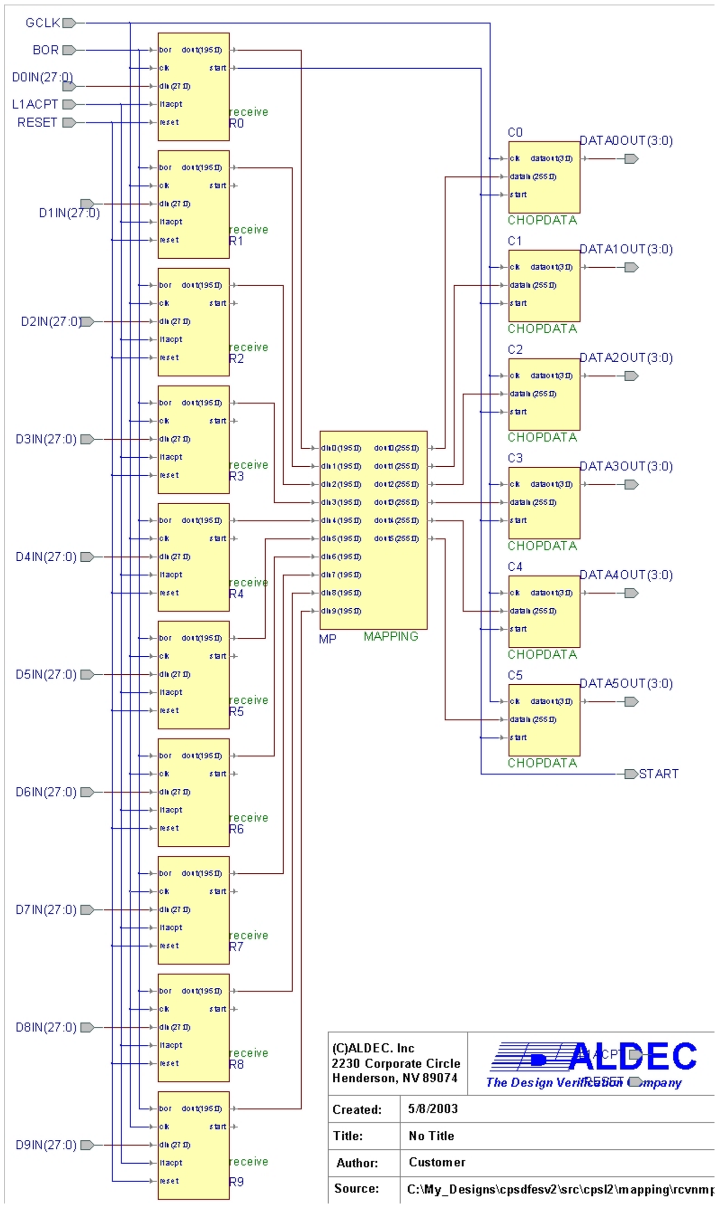


Fig. 8. Structure of RCVNMP module

The Figure 8 shows the structure of module RCVNMP(rcvnmp.vhd). It includes four sub-modules. The RECEIVE (receive.vhd) take raw data from memory of DFES_FRONT_END. Once the begin of data (BOR) and L1ACPT (Level 1 accept) are found, it change 7-frame series input into a parallel data and send data to global mapping module MAPPING (mapping.vhd). After all 10 links data in 7 frame data and mapping are ready, a signal START is sent out to the module CHOPDATA (chopdata.vhd), which begins to send out data in the width of 4. At the same time, the START also tell next module to begin the procession to search for clusters.

The details of module CLUSTER is shown in Figure 9. The incoming data with width of 4-bit goes through a look-up table (LUT) module FND_CLST (fnd_clst.vhd). The outputs are the addresses and widths of clusters. Also the corresponding bit-0 and bit 3 of that cluster is included for next stage to combine cluster, if necessary. The signals C1CLST and C2CLST indicate if cluster exists in the data. The modules of ADDRESS (address.vhd) and WIDTH (width.vhd) combine the cluster according to the signal SEL, which comes from the module SELECTOR (selector.vhd). Since the maximum width of a cluster to report is 8, the signal OVFL indicate the overflowing of cluster width, and correspondingly the width and address have to be adjusted. The cluster data are registered in OUT_REG (out_reg.vhd). The module MERGE (merge.vhd) combines the address and width into a 16-bit data and also changes the address of cluster from the beginning of cluster to the center of the cluster. The combined cluster data are stored in a dual port memory module DPMEM16X64 (dpmem16x64.vhd), which its address and read/write operation are controlled by the module MEMCTRL (memctrl.vhd). This module can also give out the total cluster number, CNUM, to the next step.

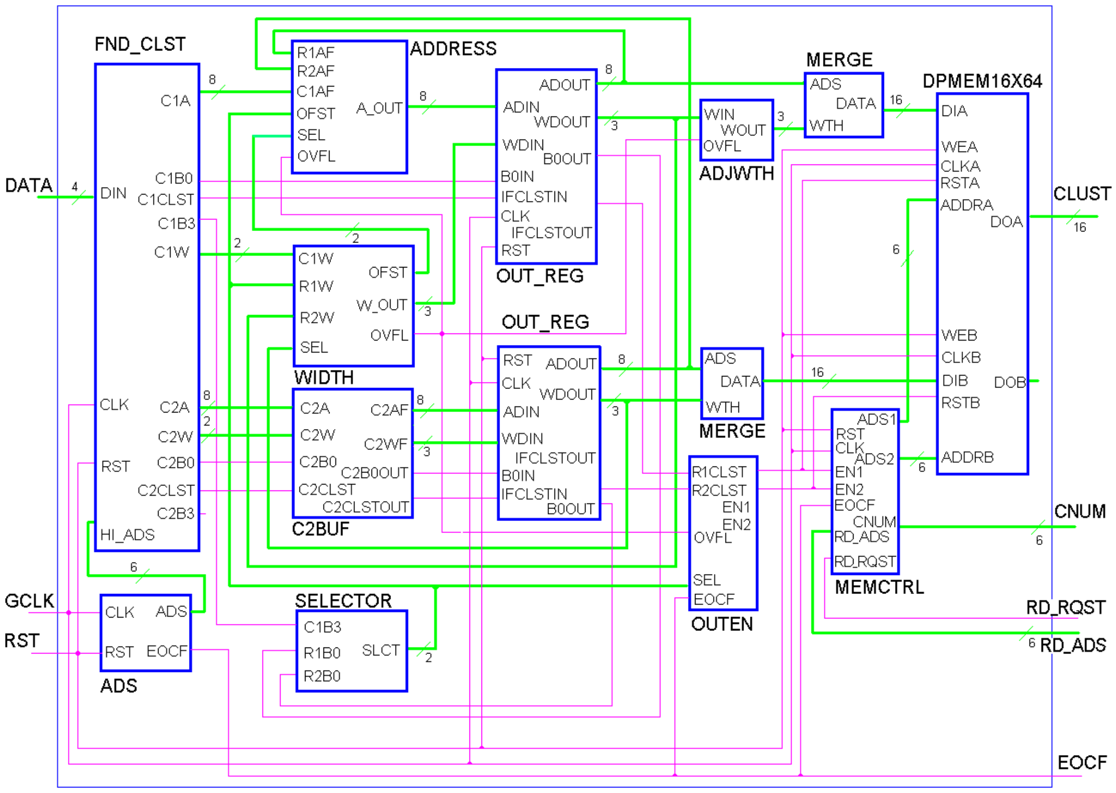


Fig. 9. Diagram of module CLUSTER

IV. Test

For test purpose, we generate a set of test vectors. Because U/V mixing, the test vectors have to be in pair so that a complete U or V super-sector can be formed. The test vectors for two links, without control bit set, are:

	frame 1	frame2	frame3	Frame4	frame5	frame6	Frame7
1st link	0001000	000005E	7000000	07CC000	0000000	0000000	0000000
2nd link	0600000	0000000	0000000	0003000	4000016	00CC000	0000000

After mapping the data become a 256-bit width data, which is, for U layer, 00BE 0000 0000 0000 0600 0000 0000 7C00, and for V layer, 0000 001F F800 0000 0000 0000 03D0 0000. The first bit of each array corresponds to the first fiber of CPS detector.

From the data, we can see that there are total 7 clusters in the data. However, one cluster has the width of 10. Since the maximum width is 8, this cluster has to be cut into two clusters. The first cluster width will be 8 and the 2nd one has the width that is left. So, if we have test vector in Link 0/1 and 2/3, and no data for the rest of links (or links not connected.), the output for this test vector, starting from the 1st cluster and without header and trailer, is 0000600, 000800, 0004600, 0000C00, 0001600, 0004500, 0004600,0007300, 0000600, 0008800, 0004600, 0008C00, 0001600, 000C500, 0004600, 000F300, 0007A00, 0001E00, 0001A00, 0002300, 0003A00, 0006700, 0000A00, 0006B00, 0007A00, 0009E00, 0001A00, 000A300, 0003A00, 000E700, 0000A00, 000EB00.

V. Results

The algorithm we present here is the core of the Cluster Finding operation for the Preshower. In addition, there are other algorithms necessary to complete the function, such as algorithms dealing with synchronization, mapping, data formatting and cluster reporting. All these are implemented with the same CAD and in the same FPGA.

In our design, data of 1280 elements are processed by five parallel implementations of the described algorithm and necessary logic to combine possible clusters. At a clock frequency of 53MHz, the whole process takes only 64 clocks, or 1.21 microseconds. Including all function blocks, this design takes about 22% of FPGA resources that are required by the implementation of the “standard” algorithm.

VI. ACKNOWLEDGMENT

We would like to thank everyone in the VHDL group at Dzero experiment at Fermilab for their continuous help and suggestion, especially Manuel Martin, Levan Babukhadia, Jamieson Olsen,Zhengguo Zhao, Makoto Tomoto, Stephan Linn, Ricardo Ramirez and Carsten Noeding.

VI. References

1. “The Dzero upgrade”, pp.36-43, Fermilab Pub-96/357-E, available at http://higgs.physics.lsa.umich.edu/dzero/d0doc96/d0doc.html
2. M. Adams, N. Amos, S. Chopra, M. Chung, K. DelSignore, F. Hsieh, *et al.*, “Design Report of the Central PreShower Detector for the D0 upgrade”, Dzero internal report, January 1996.